# Armv7-M架构

## 1 简介

### 1.1 Armv7体系结构

* Armv7-A：支持Arm和Thumb指令集以及虚拟内存管理；
* Armv7-R：支持Arm和Thumb指令集，不支持虚拟内存；
* Armv7-M：仅支持Thumb指令集的微控制器，性能相对较差但是稳定；

如上所述，Armv7主要包含这3种。

### 1.2 Armv7-M架构概要

在Armv6T2中引入Thumb-2技术为Arm和Thumb指令集提供了平衡并且为Arm架构进入微控制器市场提供了机会。Arm推出了用于微控制器实现的Armv7-M架构以增加其在高性能和实时嵌入式市场的优势。

Armv7-M特性：

* 在功耗、性能和面积上具有优势；
* 具有较强稳定性：单周期或低周期数执行；中断延时小；
* 对C/C++支持较好；函数调用按照标准进行；
* 专为嵌入式系统设计；引脚数目相对较少；
* 提供调试和软件分析支持；

### 1.3 架构扩展

* **DSP扩展：**此可选扩展将Arm数字信号处理(DSP)指令添加到Armv7-M Thumb指令集中，包括单指令流多数据流(SIMD)指令(如向量运算指令)；
* **浮点扩展**：此可选扩展将浮点指令添加到Armv7-M Thumb指令集中。浮点扩展包含FPv4-SP和FPv5指令。FPv4-SP是为Armv7-A和Armv7-R架构定义的VFPv4-D16扩展的单精度实现。FPv5则添加了对双精度计算的可选支持，并提供了额外指令；

## 2 应用层编程模型

### 2.1 应用程序编程模型

系统级要求访问体系架构的所有模块和功能，访问操作处于特权操作。系统代码决定应用程序是以特权方式运行还是以非特权方式运行。操作系统支持特权和非特权操作，但应用程序通常只能在非特权下运行。

**运行在非特权访问的应用程序：**

* 操作系统可以将系统资源作为私有或共享资源分配给应用程序；
* 为其他进程和任务提供一定程度的保护，防止故障应用程序破坏操作系统；

运行在非特权模式意味着处理器正处于线程模式。

线程模式是Armv7-M中应用程序执行的基本模式，在系统复位时默认选择。线程模式可以在非特权或特权下存在。线程模式可以使用SVC指令引发系统调用，该指令会导致系统调用异常。线程模式也可以直接处理系统访问和控制。所有异常都由相应处理程序进行处理。系统调用处理程序主要用来管理系统资源，如与外围设备的交互、内存分配和堆栈管理。

### 2.2 Arm数据类型

Armv7-M架构支持内存中字节(8位)、半字(16位)和字(32位)的数据类型。寄存器大小为32位。加载和存储操作可以将字节、半字或字传输到内存或从内存传输。字节或半字的加载会在加载数据时进行零扩展或符号扩展。

指令集包括加载和存储操作，这些操作将两个或多个字传送到存储器或从存储器传送，可以使用这些指令加载和存储64位整数。对64位整数的直接指令支持有限，大多数64位操作需要两个或多个指令序列合并执行。

指令集提供对寄存器中的值进行位逻辑操作、移位、加法、减法和乘法。

### 2.3 寄存器和执行状态

应用程序编程模型提供了应用程序可见的通用和专用寄存器、Arm内存模型以及用于从内存加载寄存器、将寄存器存储到内存或操作寄存器内数据的指令集的详细信息。

#### 2.3.1 Arm核心寄存器

具有13个通用32位寄存器R0-R12，以及另外三个具有特殊名称和使用模型的32位寄存器。

* SP：堆栈指针，用作指向活动堆栈的指针；SP有时被称为R13；
* LR：链路寄存器，用于存储返回指针。与子例程返回地址相关的值，该子例程使用带链接的分支指令输入。复位将该寄存器设置为0xFFFFFF。LR也会在进入异常时更新，LR有时被称为R14；
* PC：程序计数器，PC在复位时加载复位处理程序起始地址；PC有时被称为R15；

#### 2.3.2 程序状态寄存器(APSR）

程序运行状态在APSR中记录。保留位被分配给系统功能和用于未来扩展。

* N标志位：第31位，负条件标志。设置为指令结果的第31位。如果结果被视为2的补码有符号整数，则结果为负，则N设置为1否则设置为0；
* Z标志位： 第30位，零条件标志。如果指令结果为零，则设置为1，否则设置为0；
* C标志位：第29位，进位条件标志。如果指令导致进位产生则设置为1；

#### 2.3.3 特权执行

系统代码决定应用程序是以特权还是非特权方式运行。在一个操作系统同时提供特权和非特权支持的情况下，应用程序通常是非特权运行的。操作系统分配系统资源供应用程序私有或共享使用，并为其他进程和任务提供一定程度的保护

线程模式是Armv7-M中应用程序执行的基本模式。线程模式在重启时选择，并且可以根据系统环境以特权或非特权方式执行。当代码在非特权状态下执行时，线程模式可以执行SVC指令以生成系统调用异常。线程模式下的特权执行可以使用SVC进行系统调用或者直接处理系统访问和控制。

### 2.4 异常、故障和中断

异常可由异常生成指令引起或由系统的行为而触发。在体系结构中可能发生同步和异步异常。

#### 2.4.1 系统相关事件

应用程序代码使用系统调用从底层操作系统请求服务。使用SVC指令，应用程序可以发起对系统进行特权访问的服务的调用。

**可能发生的故障：**

* 指令执行相关错误；
* 存储器访问错误；
* 各种执行状态相关错误导致的故障；
* 调试事件可生成调试异常；

Armv7-M实现的中断包括：

* 系统定时器、SysTick和相关中断；
* 异步的系统调用，即PendSV。当处理程序需要其它系统服务时，处理程序会使用该调用。当处理器接受相关异常时，PendSV处理程序执行；
* 外部中断控制器；
* 生成调试事件的BKPT指令；

出于电源或性能原因，软件可能希望通知系统某个操作已完成或者向系统说明可以暂停当前任务。Armv7-M体系结构提供相应指令：

* 发送事件和等待事件指令，如SEV和WFE；
* 等待中断指令，如WFI；

### 2.5 可选的浮点扩展

浮点扩展是Armv7-M的可选扩展，分为FPv4-SP和FPv5两个版本。两个版本都定义了支持单精度(32位)运算的浮点单元(FPU)，而FPv5还提供了额外的指令，并可选地支持双精度(64位)运算。

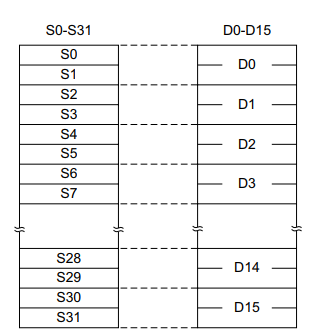
**FPv4-SP FPU支持：**

* FP扩展寄存器，软件可以将其视为32个单精度或16个双精度寄存器；
* 单精度浮点运算；
* 整数、单精度浮点和半精度浮点格式之间的转换；
* 单精度和双精度寄存器的数据传输；

FPv5 FPU包括FPv4 SP的所有功能，并添加了可选的双精度浮点运算以及整数、单精度浮点、双精度浮点和半精度浮点格式之间的转换。

#### 2.5.1 FP扩展寄存器

软件可以访问FP扩展寄存器组：32位单精度寄存器(S0-S31)和16个64位双精度寄存器(D0-D15)。



该扩展可以同时使用这两个视图。复位后，FP扩展寄存器的值为未知的。在保存FP上下文之后，寄存器S0-S15的值是未知的。保存FP上下文不会保存寄存器S16-S31的值，也不会影响这些寄存器的值。

FP扩展提供在寄存器S0-S31上操作的单精度浮点数据处理指令和在寄存器D0-D15上操作的双精度浮点数据处理指令(可选的)。

#### 2.5.2 浮点状态控制寄存器(FPSCR)

该寄存器提供浮点系统的应用程序级控制，32位可读写寄存器，可进行非特权和特权访问。FPSCR复位值未知。创建新的浮点上下文时需要将FPSCR的AHP、DN、FZ和RMode字段设置为FPDSCR中指定的值。

* N标志位：第31]位，负条件标志。浮点比较操作更新此标志；
* Z标志位：第30位，零条件标志。浮点比较操作更新此标志；
* C标志位：第29位，进位条件标志。浮点比较操作更新此标志；
* V标志位：第28位，溢出条件标志。浮点比较操作更新此标志；

### 2.6 协处理器支持

Armv7-M可以选择性支持协处理器。如果不支持则将所有协处理器视为不存在。协处理器编号从0到15，称为CP0-CP15。Arm保留CP8到CP15，CP0到CP7是根据协处理器指令的约束实现。

协处理器10和11提供浮点运算的Armv7-M浮点(FP)扩展。在包含FP扩展的Armv7-M实现中，软件必须启用对CP10和CP11的访问，然后才能使用扩展的任何功能。

如果软件向不存在或禁用的协处理器发出协处理器指令，处理器将生成没有相应协处理器的异常。如果软件向已启用的协处理器发出未知指令，处理器将生成未知指令异常。

## 3 Arm架构内存模型

### 3.1 地址空间

Armv7-M体系结构支持4GB内存空间。字节地址为从0到0xFFFFFFFF。每个32位字的地址都是字对齐的，这意味着地址可被4整除。每个16位半字都是半字对齐的，这意味着地址可被2整除。

指令存储的地址总是半字对齐的，但一些加载和存储指令支持未对齐的地址。下一条指令地址为当前指令地址增加2或4。如果计算的指令地址溢出了地址空间的顶部，结果是不可预测的。

Armv7-M中使用的所有内存地址都是物理地址，不支持虚拟地址。对于Armv7-M架构配置，在所有情况下，修改的虚拟地址(MVA)、虚拟地址(VA)和物理地址(PA)具有相同的值。

### 3.2 对齐支持

Armv7-M中的对齐检查包含两种策略分别为支持未对齐访问和发生未对齐访问时产生故障。策略因访问类型而异。实现可以将为不支持非对齐访问，即非对齐访问会引起故障。

**产生非对齐故障**：

* 非半字对齐LDREXH和STREXH；
* 非字对齐LDREX和STREX；
* 非字对齐LDRD、LDMIA、LDMDB、POP、LDC、VLDR、VLDM和VPOP；
* 非字对齐的STRD、STMIA、STMDB、PUSH、STC、VSTR、VSTM和VPUSH；

以下数据访问支持非对齐访问，仅在CCR寄存器中UNALIGN\_TRP位设置为1时生成非对齐故障：

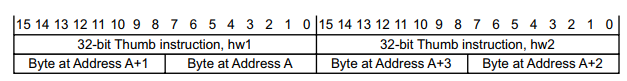
* 非半字对齐TBH；
* 非字对齐的LDR和STR；

所有指令地址必须半字对齐。

### 3.3 大小端支持

Armv7-M支持可选择的大小端。在重置时，控制输入确定大端序还是小端序。设置仅适用于数据访问，指令总是小端序的。所有对SCS的访问均为小端序。AIRCR寄存器中ENDIANNESS表示数据端序。

Thumb指令执行时对所有指令强制16位对齐。这意味着32位指令被视为两个半字hw1和hw2，hw1位于较低地址。



当应用程序或设备驱动程序必须与内存映射的外围寄存器或共享内存进行数据交换时，这些内存映射的外部寄存器或共享的内存与内部数据结构或操作系统的端序不同，需要一种能够显式转换数据端性的有效方法。Armv7-M支持以下字节转换指令：REV(变换32位)，REVSH(变换带符合16位)和REV16(变换无符号16位)。

### 3.4 同步和信号量

独占访问指令支持非阻塞共享内存访问同步，允许在读写阶段之间对信号量进行计算，并扩展多处理器系统设计。

在Armv7-M中，提供的同步原语为：

* 独占加载：LDREX、LDREXB和LDREXH；
* 独占存储：STREX、STREXB和STREXH；
* 独占清除：CLREX；

每个独占加载指令只能与相应独占存储指令一起使用。在Armv7-M中不支持STREXD和LDREXD。

使用独占加载存储指令访问内存：

* 独占加载指令总是能够从内存地址成功读取值；
* 只有当没有其他处理器或进程执行了对相同地址的写入时，对应的独占存储指令才能成功写入内存。该操作返回一个状态位，表示内存写入是否成功；

独占加载指令标记一小块内存以进行独占访问。同一地址的独占存储指令清除标记。

#### 3.4.1 独占访问和不可共享内存区域

对于不具有可共享属性的内存区域，独占访问指令依赖于本地监视器，该监视器标记处理器执行独占加载的任何地址。同一处理器使用独占存储指令来清除该地址的标记。

独占加载指令执行从内存加载并且标记物理内存以进行断站访问，然后执行该访问处理器的本地监视器转换到其独占访问状态。

独占存储指令执行对内存的条件存储，这取决于本地监视器的状态。

如果本地监视器处于独占访问状态且存储同步的地址与先前加载同步在监视器中标记的地址相同，则存储发生，否则由具体实现定义存储是否发生。然后根据存储是否发生返回0或1；本地监视器会转为开放访问状态。

如果本地监视器处于开放访问状态，独占加载指令不会导致存储发生；状态寄存器始终返回1；本地监视器继续保持开放访问状态。

#### 3.4.2 独占访问和可共享内存区域

系统中每个处理器的本地监视器会标记处理器执行独占加载的任何地址。除了可共享内存之外，更新内存和返回状态值0中的任何一个或两个的独占存储都要接受全局监视器的检查。本地监视器可以忽略来自系统中其他处理器的独占访问。

全局监视器将物理地址标记为特定处理器的独占访问。此标记稍后用于确定本地监视器允许的独占存储是否会发生。任何其他观察者对共享性内存中被标记地址的任何成功写入都会清除独占访问标记。对于系统中的每个处理器，全局监视器只保存单个标记地址。

全局监视器可以在处理器中，也可以作为内存接口中的辅助监视器存在。实现可以将全局监视器和本地监视器的功能组合到一个单元中。

全局监视器从共享内存中独占加载的操作执行内存加载，并使访问的物理地址被标记为请求处理器的独占访问。这种访问还导致从请求处理器标记的任何其他物理地址中移除独占访问标记。全局监视器仅支持一个共享内存独占访问标记。

存储独占执行对内存的条件存储：

* 只有当访问的物理地址被标记为请求处理器的独占访问并且请求处理器的本地监视器和全局监视器状态机都处于独占访问状态时，才能保证存储成功；请求处理器的全局监视器状态机的最终状态是实现定义的；如果访问的地址在全局监视器状态机中标记为任何其他处理器的独占访问，则该状态机转换为开放访问状态；
* 如果没有地址被标记为请求处理器的独占访问，则存储不成功；全局监视器不受影响，并保持请求处理器的开放访问状态；
* 如果不同的物理地址被标记为请求处理器的独占访问，则定义存储是否成功的实现；如果处理器的全局监视器状态机在存储独占之前处于独占访问状态，则定义该状态机是否转换为开放访问状态；

存系统中，全局监视器为系统中的每个处理器实现单独的状态机。处理器n访问共享存储器的状态机可以响应对其可见的所有可共享存储器访问。共享内存系统中，全局监视器为每个观察者实现了一个单独的状态机，该状态机可以在系统中生成独占加载或独占存储。

#### 3.4.3 标记和标记内存块的大小

当执行LDREX指令时，生成的标记地址忽略内存地址的最低有效位。该分配中的最低有效位是在最小值2和最大值11之间。将内存分为相应块进行标记。

#### 3.4.4 上下文切换支持

必须确保本地监视器在上下文切换后处于开放访问状态。在Armv7-M中，当进入或退出异常时，本地监视器自动转为开放访问状态。本地监视器也可以通过CLREX指令强制为开放访问状态。上下文切换可能导致后续独占存储失败，需要重新进行加载存储。为了最大限度地降低这种情况发生的可能性，Arm建议存储独占指令尽可能靠近相关的加载独占指令。

### 3.5 内存类型和属性

#### 3.5.1 内存类型

有三种互斥的内存类型普通类型、设备类型和强有序类型。普通和设备存储器区域具有附加属性。通常，用于程序代码和数据存储的内存是普通内存。

普通存储具有可编程闪存ROM、ROM、SRAM、DRAM和DDR存储器。

系统外设通常符合对正常存储器的不同访问规则。I/O访问的内存区域包含具有连续访问的FIFO、中断控制器寄存器、内存控制器配置寄存器和内存映射外设。

为确保系统正确性，设备和强顺序内存的访问规则比普通内存的访问限制更严格。此外对于强有序存储器，所有存储器访问都严格按照访问存储器指令的顺序进行。

**强有序存储器访问**：

* 读和写访问都有副作用；
* 访问不能重复，例如从异常返回时；
* 必须维护访问的数量、顺序和大小；

#### 3.5.2 Arm体系结构中的原子性

原子性是内存访问的一个特性，称为原子访问。Arm体系结构描述涉及两种类型的原子性分别为单拷贝原子性和多拷贝原子性。

**单拷贝原子性**：

* 在对一个操作数进行任意多次写操作之后，该操作数的值就是其中一次写操作所写的值。不可能出现不同部分值来源于不同写操作；即写操作之间不会被打断；
* 当对同一个操作数进行读操作和写操作时，读操作的值部分是写操作之前的操作数的值，部分是写操作之后的操作数的值，这种情况永远不会发生；

在Armv7-M中，单拷贝原子访问包括所有字节访问、所有对半字对齐地址的半字访问和所有对字齐位置的字访问。

当不是单拷贝原子访问时，将拆分为一系列较小的访问执行，其中每个访问都是单拷贝原子访问，至少在字节级别上是这样。

**多拷贝原子访问**：

* 对同一个位置的所有写操作都是序列化的；这意味着所有的观察者都会观察到相同的操作顺序，尽管有些观察者可能不会观察到所有的写操作；
* 对一个位置的读操作不会返回写操作的值，直到所有的观察者都观察到该写操作；

对普通内存的写入不是多拷贝原子性的。对设备和强顺序内存的所有写操作都是单拷贝原子性，也是多拷贝原子性。对同一位置的所有写操作都是序列化的。对普通内存的写操作可以重复进行，直到观察到对同一地址的另一次写为止。对于普通内存，写操作的序列化并不禁止写操作的合并。

#### 3.5.3 普通内存

正常内存是幂等的，这意味着具有以下属性：

* 读取访问可以重复，没有副作用；
* 重复读取访问返回写入正在读取的存储单元的最后一个值；
* 读访问可以预取额外的内存位置，无副作用；
* 只要存储内容在重复写入之间保持不变，则可以重复写入访问而不会产生副作用；
* 可支持非对齐访问；
* 在访问目标内存系统之前，可以合并访问；

普通内存可以是读写或只读的，普通内存区域定义为可共享或不可共享。对正常内存的访问具有弱一致的内存排序模型。当必须控制其他观察者观察到的存储器访问顺序时，需要屏障操作。无论正常内存区域的可缓存性和可共享性属性如何，此要求都适用。

非共享属性标识可能仅由单个处理器访问的普通内存。标记为不可共享正常的内存区域不需要使缓存的效果对数据或指令访问透明。如果其他观察者共享内存系统，如果缓存的存在可能导致观察者之间通信时的一致性问题，则软件必须使用缓存维护操作。此缓存维护要求是对确保内存排序所需的屏障操作的补充。对于不可共享的普通内存，加载独占和存储独占同步原语不考虑多个观察者访问的可能性。

可共享内存属性描述了预期由多个处理器或其他系统请求者访问的普通内存。具有可共享属性的普通内存区域是在内存系统上插入一个或多个缓存的效果对同一共享性域中的数据访问完全透明的区域。需要明确的软件管理来确保指令缓存的一致性。对于可共享普通内存，加载独占和存储独占同步原语考虑了同一可共享域中多个观察者访问的可能性。

除了可共享或不可共享之外，正常内存的每个区域都可以标记为可写可缓存、回写可缓存以及不可缓存。区域的可缓存性属性独立于区域的可共享性属性。如果数据区域用于处理共享数据以外的其他目的，则可缓存性属性指示所需的数据区域处理。

#### 3.5.4 设备存储器

设备内存类型定义了内存位置，其中对该内存的访问可能会导致副作用或加载返回的值可能会根据执行的加载数量而变化。内存映射外围设备和I/O内存是通常标记为设备内存区域。

从处理器到标记为设备内存的显式访问都以其程序大小进行并且访问次数是程序指定的次数。

如果程序对设备内存位置只有一次访问，则实现不得重复对该位置的访问。对设备存储器的访问是不可重复的。该架构不允许对标记为设备的内存进行预访问。标记为设备的地址位置不可缓存。

对同一地址的多次访问不得改变对该地址的访问次数。对设备存储器的访问不允许合并。当设备内存操作具有适用于正常内存区域的副作用时，软件必须使用内存屏障以确保正确执行。

设备存储器区域的可共享属性设备存储器区域可以被赋予可共享属性。这意味着设备内存区域可以描述为可共享设备内存和不可共享设备内存。不可共享设备内存定义为仅可由单个处理器访问。

#### 3.5.5 强序存储器

强排序内存类型定义了内存位置，其中对该内存的访问可能会导致副作用或者加载返回的值可能会根据执行的加载数量而变化。

对于从处理器到内存的显式访问，当所有访问都以其程序大小发生或访问次数是程序指定的次数则标记为强有序访问。

实现对强有序内存的访问次数不得超过程序的简单顺序执行所指定的次数，除非异常。

该体系结构不允许对标记为强有序的内存进行数据预访问。

强有序内存中的地址对应位置内容不保存在缓存中，始终被视为可共享内存位置。

对于生成访问序列的指令，可能会因为在访问序列期间发生异常而被放弃。在从异常返回时，指令会重新启动，因此一个或多个内存位置可能会被多次访问。这可能导致对一个位置的重复写访问，而该位置在写访问之间已经改变。

任何对强顺序内存的未对齐访问都会导致不可预知的行为。

#### 3.5.6 内存访问限制

* 跨越不同内存类型之间边界的未对齐访问是不可预测的；
* 对于由同一指令生成的任何两个内存访问X和Y，X和Y访问的字节必须都具有相同的内存类型属性，否则结果是不可预测的；
* 生成对设备或强顺序内存的未对齐内存访问的指令是不可预测的；
* 对于生成对设备或强序内存访问的指令，实现不得更改指令伪代码指定的访问顺序；

### 3.6 访问权限

Armv7包括内存区域的附加属性。这些属性包含根据访问权限限制数据访问和根据进行获取的进程或线程的权限限制指令获取。

#### 3.6.1 数据访问的特权级别访问

内存属性可以定义内存无法进行访问、仅允许特权访问或可进行特权和非特权访问。访问权限级别分为显式读访问和显式写访问。定义存储器属性的系统不需要支持用于读和写访问的存储器属性的所有组合。

特权访问是在特权执行期间，由于加载或存储操作而进行的访问而不是由LDRT、STRT、LDRT、STRT、LDRT、LDRSHT或LDRSBT指令引起的访问。

**非特权访问**：

* 当前执行模式配置为非特权访问时；
* 当处理器处于任何模式，并且由于LDRT, STRT, LDRBT, STRBT, LDRHT而进行访问时；

如果处理器试图进行访问权限不允许的数据访问，则会发生异常。

#### 3.6.2 指令访问的特权级别访问控制

内存属性可以定义内存区域为不可执行、仅可由特权进程执行或可由特权和非特权进程执行。

为了定义对内存区域的指令访问权限，内存属性分别描述该区域：其读取访问权限和区域是否为从不执行。这意味着定义区域对显式内存访问的内存属性与定义区域可以执行的内存属性之间存在某种联系。

### 3.7 内存访问顺序

Armv7提供了三种内存类型：普通内存、设备内存和强有序内存，具有定义良好的内存访问属性。

在考虑内存访问排序时，一个重要的特性是可共享内存属性，该属性指示一个内存区域是否可以在多个处理器之间共享，因此需要排序模型中出现缓存透明性。

在应用程序对于普通内存的访问，在某些情况下，必须通过设置内存屏障来控制其他观察者观察到的访问顺序。

对于芯片设计者需要考虑到系统级的模型，强有序和设备内存属性对系统设计者何时指示访问完成方面施加了某些限制。

#### 3.7.1 读写操作

每次内存访问都是读或写。显式内存访问是指令功能所需的内存访问。指令获取以及缓存加载和回写不属于显示的内存访问。除非另有说明，内存排序只适用于显式内存访问。

同步原语必须确保内存顺序模型中正确操作系统信号量。可共享和不可共享内存支持加载独占和存储独占指令。

非共享内存可用于同步在同一处理器上运行的进程。共享内存必须用于同步可能在不同处理器上运行的进程。可观察性和完整性可以观察内存访问的观察者集由系统定义。

对于所有内存：

* 当同一个观察者随后读取内存中的某个位置时，该观察者将返回写入时写入的值；
* 当可共享性域内的任何观察者对位置的后续读取将返回由写入者写入的值时，对存储器中位置的写入被称为全局观察；
* 当同一观察者对存储器中某个位置的后续写入不会对读取返回的值产生影响时，观察者可以观察到存储器中某一位置的读取；
* 当可共享性域内的任何观察者对位置的后续写入不会对读取返回的值产生影响时，存储器中该位置的读取被称为可共享性区域的全局观察；

强有序或设备存储器中的存储器访问的完成不能保证存储器访问的副作用对所有观察者可见。确保内存访问副作用可见性的机制是实现定义的。

#### 3.7.2 内存访问的排序要求

Armv7-M在允许的内存访问顺序中定义了访问限制。这些限制取决于所涉及访问的内存属性。

* **地址依赖性**：当读访问返回的值用于计算后续读或写访问的地址时，存在地址依赖性。即使由第一读取访问读取的值不改变第二读取或写入访问的地址，也存在地址依赖性。如果返回的值在使用前被屏蔽，或者对第二次访问的预测地址值没有影响，则可能会出现这种情况；
* **控制依赖性**：当读访问返回的数据值用于确定条件并且条件会影响后续读访问的地址时，存在控制依赖性。该地址确定可以通过条件执行，也可以通过对分支的求值；

指令执行的程序顺序是控制流跟踪中指令的顺序。

#### 3.7.3 内存屏障

内存屏障是应用于指令或指令序列的通用术语，用于强制处理器执行同步事件。

内存屏障用于保证：

* 完成预加载和存储指令；
* 在内存屏障事件之前刷新任何预取指令；

Armv7-M需要六个显式内存屏障来支持本章中描述的内存顺序模型。

* 推测数据屏障(CSDB)；
* 数据存储屏障(DMB)；
* 数据同步屏障(DSB)；
* 指令同步屏障(ISB)；
* 物理推测存储旁路屏障(PSSBB)；
* 推测性存储旁路屏障(SSBB)；

DMB和DSB内存屏障影响加载和存储指令生成的对内存系统的读写。指令获取不是显式访问，不受影响。

推测性数据屏障(CSDB)中 CSDB指令是一种内存屏障，防止在数据屏障完成后以程序顺序出现的指令确定从推测性执行的加载指令中导出的数据的任何部分，这些加载指令在CSDB指令完成前以程序顺序显示。

DMB指令是一种数据内存屏障。执行DMB指令的处理器称为执行处理器Pe。DMB指令将可共享性域和访问类型作为参数。DMB创建将内存访问分为A组和B组。

**组A访问**：

* 在执行DMB指令之前，来自任何与Pe在同一个共享域的所有观察者对相应访问类型的显式内存访问；
* 任何来自与Pe具有相同共享域的相应加载访问类型被组A成员Py且Py与Pe具有系统共享域且在Py进行内存访问之前；

**组B访问**：

* 在DMB指令之后，所有需要访问类型的显式内存访问都是由Pe按指令顺序进行；
* 在与Pe相同的共享域内，任何给定的观察者Px进行所需访问类型的显式内存访问时只有在Px观察到组B成员的存储后才会发生；

任何与Pe具有相同共享域的观察者都会先观察组A的所有成员，然后再观察组B的任何成员。当组A和组B的成员访问相同的内存映射外设时，组A比组B的所有成员在内存映射外设上较先可见。DMB只影响内存访问，对处理器上执行的任何其他指令的顺序没有影响。

DSB指令是一种特殊的内存屏障，使执行流与内存访问同步。DSB指令将可共享性域和访问类型作为参数。当与DMB具有相同的参数时DSB表现相同并且还具有此处定义的附加属性。

DSB指令完成：

* 在执行DSB之前，Pe观察到的所有显式内存访问，都是必需的访问类型并且来自Pe所要求的共享域内的观察者，对于所要求的共享域内的一组观察者来说都是完整的；
* 在DSB完成之前，所有对系统控制空间(SCS)的显式访问导致Pe发出的上下文改变操作；

此外，在DSB指令完成之前，程序中出现的任何指令都不能执行。

ISB指令会刷新处理器中的流水线。因此，只有在ISB指令完成后，才会从高速缓存或内存中获取排在ISB指令之后的指令。使用ISB确保在ISB之前执行的上下文更改操作的效果对于ISB指令之后获取的指令是可见的。

需要插入ISB指令的例子：

* 确保系统控制更新；
* 对具有可配置优先级的异常重新排序；

此外，在ISB指令之后出现的任何分支都被写入带有ISB指令之后可见的上下文的分支预测逻辑中。这是确保指令流正确执行所必需的。在ISB指令之后出现的任何改变上下文的操作只有在ISB指令被执行后才会生效。

Armv7-M实现必须选择在当前执行点之前多远预取指令。这可以是固定数量的指令，也可以是动态变化的指令。除了选择要预取多少条指令之外，实现还可以选择未来可能的执行路径，即分支预测。

所有形式的指令预取都存在一个潜在的问题，即内存中的指令可能在预取之后但在执行之前被更改。如果发生这种情况，对内存中指令的修改通常不会阻止已经预取的指令副本执行到完成。内存屏障指令(ISB、DMB或DSB)在必要时用于强制执行排序。

**PSSBB指令防止预加载**：

* 在加载前，返回更旧的数据而不是最近加载到相同物理地址的数据；
* 在加载后返回相同物理地址存储的数据；

**SSBB指令防止预加载**：

* 将比最近的存储更早的数据返回到加载前出现在程序顺序中的相同虚拟地址；
* 使用相同的虚拟地址从存储返回数据，在加载后显示程序顺序；

该体系结构将SCS定义为强有序内存。除了强有序内存的行为规则外，该体系结构还要求在访问完成时，对执行上下文更改操作的SCS的任何访问的副作用生效。软件可以发出DSB指令，以保证完成先前的SCS访问。

该体系结构保证了上下文更改操作效果的可见性，仅适用于执行上下文更改操作的SCS访问完成后获取的指令。执行ISB指令，或者执行异常项或异常返回，都可以保证重新获取已经获取但没有执行的任何指令。

为了保证先前SCS访问的副作用是可见的，软件可以执行DSB指令，然后执行ISB指令。

### 3.8 缓存和内存层次结构

Armv7-M通过内存属性定义了对架构内缓存的支持。可以在支持总线协议上导出内存属性，以支持系统缓存。在可能出现一致性问题的情况下，软件必须使用缓存维护指令来管理缓存，这些操作是内存映射和实现定义的。

#### 3.8.1 缓存简介

高速缓存是包含地址信息和相关数据的高速存储单元块。目的是提高存储器访问的平均速度。高速缓存基于两个局部性原则：空间局部性和时间局部性。

为了最小化存储的控制信息的数量，使用空间位置属性将多个位置分组在同一标签下。该逻辑块通常称为缓存线。当数据加载到缓存中时，后续加载和存储的访问时间会减少，从而带来整体性能优势。对缓存中已有信息的访问称为缓存命中，其他访问称为高速缓存未命中。

每当处理器想要访问可缓存位置时，都会检查缓存。如果访问是缓存命中，则访问将立即发生，否则将分配一个位置并从内存加载缓存行。不同的缓存拓扑和访问策略是可能的，但必须符合底层体系结构的内存一致性模型。

#### 3.8.2 内存层次结构

靠近处理器的内存具有非常低的延迟，但大小有限，实现成本昂贵。离处理器更远的地方，更容易实现更大的内存块，但这些内存块的延迟增加了。为了优化整体性能，Armv7内存系统可以在分层内存系统中包括多级缓存。

#### 3.8.3 缓存对应用程序的影响

缓存对应用程序程序员来说基本上是不可见的，但由于一致性问题，缓存可能会变得可见。在哈佛高速缓存体系结构中，当新指令数据写入数据高速缓存和内存中，但指令高速缓存仍包含旧指令数据时，会发生一致性问题。

确保缓存的数据一致性：

* 在可能出现一致性问题的情况下不使用缓存；
* 通过使用缓存维护指令来管理软件中的一致性问题；
* 通过使用硬件一致性机制，确保不同可共享性域内的观察者对可缓存位置内存的数据访问的一致性；这些硬件一致性机制的性能高度特定于实现；

指令一致性问题是从定义的实现中获取指令的时间比当前执行点提前了多远。这种预取可以是固定或动态变化数量的指令，并且可以遵循任何或所有可能的未来执行路径。

#### 3.8.4 预加载缓存

Arm架构提供存储器系统PLD(预加载数据)和PLI(预加载指令)以允许软件将存储器位置的预期使用传达给硬件。如果存储器访问发生，则存储器系统可以通过采取期望加速存储器访问的动作来响应。这些由实现定义。

预加载指令是提示，因此实现可以将它们视为NOP，而不影响设备的功能行为。指令不会生成异常，但内存系统操作可能会由于内存访问而生成异常。

## 4 Armv7-M指令集

### 4.1 指令集简介

Armv7-M支持Thumb-2技术引入Thumb指令集的大量32位指令。许多可用功能与Armv6T2和其他Armv7配置文件中的Thumb指令集支持的Arm指令集相同。Thumb指令为16位或32位，并在两字节边界上对齐。16位和32位指令可以自由混合。使用16位指令最有效地执行许多常见操作。

然而大多数16位指令只能访问8个通用寄存器，即R0-R7。这些被称为低寄存器。少量16位指令可以访问高寄存器R8-R15并且许多需要两个或更多16位指令的操作可以使用单个32位指令更有效地执行。

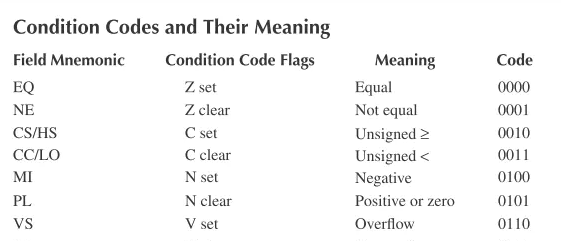
#### 4.1.1 Armv7-M和互通支持

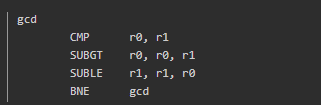
Thumb指令中地址第0位设置为1被称为互通地址。互通地址在BX或BLX以及加载PC的LDR或LDM指令中使用。

Armv7-M仅支持Thumb指令执行状态。当EPSR寄存器T标志位为0会导致异常。因此在共享指令中地址第0位必须为1，否则将发生异常。

#### 4.1.2 条件执行

条件执行意味着只有当APSR中的N、Z、C和V标志满足指令中指定的条件时才会执行该条指令。如果标志不满足此条件，则该指令将作为NOP，也就是说，执行将正常前进到下一条指令，包括对所采取的异常进行任何相关检查。大多数Thumb指令是无条件的。





如图，“suble”当r0小于r1时指令才会执行，这相对于使用分支指令而言数目减少且执行速度更快。

IT (If-Then) 指令由四条后续条件指令（IT 块）句组成。 这些条件可以完全相同，也可以互为逻辑反。IT 块中的指令（包括跳转）还必须在语法的 {cond} 部分中指定条件。无需在代码中编写 IT 指令，因为汇编器会根据在后续指令中指定的条件为您自动生成这些指令。不过，如果确实需要编写 IT 指令，则汇编器会根据后续指令中指定的条件对 IT 中指定的条件进行验证。

**语法：IT{x{y{z}}} {cond}**

* x：指定 IT 块中第二个指令的条件开关；
* y：指定 IT 块中第三个指令的条件开关；
* z：指定 IT 块中第四个指令的条件开关；
* cond：指定 IT 块中第一个指令的条件；

条件开关为T(将条件 cond 应用于指令)和E(将 cond 的相反条件应用于指令)。

IT指令不影响条件标志。不允许对IT块中的任何指令进行分支，除了由异常返回执行的分支。除CMP、CMN和TST外，IT块中的16位指令不设置条件标志。

### 4.2 统一汇编语言(UAL)

本文档使用Arm统一汇编语言（UAL）。这种汇编语言语法为所有Arm和Thumb指令提供了标准形式。UAL描述每个指令的助记符和操作数的语法。

UAL包括指令选择规则，该规则指定当多个指令可以提供所需功能时选择哪个指令编码。例如ADD指令存在16位和32位编码。最常见的指令选择规则是，当16位编码和32位编码都可用时，选择16位编码，以优化代码密度。

#### 4.2.1 条件指令

在Arm和Thumb指令集之间实现UAL汇编语言的最大可移植性：

* 在条件指令之前以Thumb命令集的正确方式编写IT指令；
* 当汇编到Arm指令集时，汇编程序会检查任何IT指令是否正确，但不会为它们生成任何代码。

尽管其他Thumb指令是无条件的，但由IT指令设置条件的所有指令都必须使用条件编写。这些条件必须与IT指令规定的条件相匹配。

某些指令不能由IT指令设置条件。如果某些指令是IT块中的最后一条指令，则这些指令可以是有条件的，但在其他情况下则不是。

包含条件字段的分支指令编码不能由IT指令设置条件。如果汇编器语法指示正确匹配前一条IT指令的条件分支，则使用不包含条件字段的分支指令编码对其进行组装。

#### 4.2.2 标签在UAL指令语法中的使用

某些指令的UAL语法包括指令或文字数据项的标签，该标签与指定的指令具有固定偏移量。

**汇编程序必须：**

* 计算指令的PC(4字节对齐)值：指令的PC值是其地址加上Thumb指令的4；
* 计算从指令的PC(4字节对齐)值到标记的指令或文字数据项的地址的偏移；
* 组装指令的PC相对编码，即读取其PC(4字节对齐)值并将计算出的偏移量相加以形成所需的地址；

### 4.3 分支指令

Thumb指令集中的分支指令。除了提供执行流的更改之外，一些分支指令还可以更改指令集。LDR、LDM和POP指令也可能导致分支。

### 4.4 数据处理指令

### 4.5 状态寄存器访问指令

MRS和MSR指令将应用程序状态寄存器(APSR)的内容移入或移出通用寄存器。APSR中的条件标志通常通过执行数据处理指令来设置并且通常用于控制条件指令的执行。可以使用MSR指令显式设置标志，也可以使用MRS指令显式读取标志的当前状态。

### 4.6 加载和存储指令

LDR指令可用于将值加载到PC中。加载的值被视为互通地址。

半字存储和字节存储分别将寄存器中的最低有效半字或字节存储到16位或8位存储器中。有符号和无符号存储之间没有区别。半字和字节加载将16或8位从存储器加载到寄存器的最低有效半字或字节中。无符号加载零将加载的值扩展到32位，有符号加载符号将值扩展为32位。

非特权加载和存储在非特权模式下，非特权加载与存储的操作方式与相应的普通操作完全相同。在特权模式下，非特权加载和存储被视为在非特权模式下执行。

加载独占和存储独占提供共享内存同步。

加载或存储的地址由两部分组成：来自基寄存器的值和偏移量。基本寄存器可以是任何一个通用寄存器。

对于加载，基本寄存器可以是PC。这允许代码无关的PC相对寻址。

**偏移量采用以下三种格式**：

* 立即数偏移量是一个无符号数，可以添加到基寄存器值或从基寄存器值中减去；
* 寄存器偏移量是来自通用寄存器的值。该寄存器不能是PC。该值可以添加到基本寄存器值，也可以从基本寄存器值中减去；
* 缩放寄存器偏移量是一个通用寄存器，而不是PC，移位一个立即值，然后与基寄存器相加或相减；

**寻址模式**：

* 将偏移量添加到基寄存器或从基寄存器中减去，以形成存储器地址；
* 将偏移量添加到基寄存器或从基寄存器中减去，以形成存储器地址。然后用这个新地址更新基寄存器，以允许通过数组或内存块进行自动索引；
* 仅将基寄存器的值用作内存地址。然后将偏移量加到基寄存器或从基寄存器中减去并且该值被存储回基寄存器中，以允许通过阵列或存储块进行自动索引；

### 4.7 加载多条和存储多条指令

加载多条指令从内存中加载通用寄存器的一个子集或可能全部。存储多条指令将通用寄存器的一个子集或可能全部存储到内存中。存储器地址是连续的字对齐地址。使用的地址从基寄存器中获得，可以高于或低于基寄存器中的值。可以根据传输的数据的总大小来更新基寄存器。

### 4.8 其他指令

### 4.9 异常生成指令

以下指令专门用于触发处理器异常：

* 监控器调用指令SVC导致SVCall异常发生。这是非特权代码调用特权操作系统代码的主要机制；
* 断点BKPT指令提供软件断点；根据调试配置，可以生成调试异常或导致正在运行的系统停止；

### 4.10 协处理器指令

有三种类型的指令用于与协处理器通信：

* 启动协处理器数据处理操作，例如 CDP、CDP2；
* 在协处理器寄存器之间传输通用寄存器，例如 MCR、MCR2、MCRR、MCRR2等指令；
* 为协处理器加载和存储指令生成地址，例如 LDC、LDC2、STC、STC2；

指令集通过每个协处理器指令中的4位字段区分多达16个协处理器，因此每个协处理器都分配了一个特定的编号。

如果Armv7-M实现包括可选的FP扩展，则它使用协处理器10和11一起提供浮点功能。扩展提供了访问这些协处理器的不同指令。这些指令与其他协处理器的指令类型类似。

## 5 Thumb指令集编码

### 5.1 Thumb指令集编码

Thumb指令流是半字对齐的半字序列。每个Thumb指令要么是该流中的单个16位半字，要么是由该流中两个连续半字组成的32位指令。

如果被解码的半字的第[15:11]位取11101、11110和11111则该半字是32位指令的第一个半字，否则半字是16位指令。

#### 5.1.1 未定义指令集空间

尝试执行未分配的指令会导致以下结果：

* 不可预测的行为。该指令被描述为不可预测；
* 未定义的指令异常。该指令被描述为未定义；

不可预测指令：

* 指令编码图中标记为（0）或（1）的位分别不是0或1，并且该编码的伪代码不表示适用不同的特殊情况；
* 在指令说明或本章中，它被声明为不可预测。

#### 5.1.2 使用1111作为寄存器说明符

在Thumb指令中通常不允许使用1111作为寄存器说明符。当允许值为1111时，可能有多种含义。

对于寄存器读取：

* 读取PC值，即当前指令的地址+4。表分支指令TBB和TBH的基寄存器可以是PC。这使得分支表可以在指令之后立即放入内存；
* 读取与字对齐的PC值，即当前指令的地址+4，位[1:0]强制为零。LDC、LDR、LDRB、LDRD等指令的基寄存器可以是字对齐PC。这可以实现PC相对数据寻址；
* 读数为零；在某些情况下，当一条指令是另一条更一般的指令的特例，但一个操作数为零时，可以执行此操作；

对于寄存器写入：

* 可将PC指定为LDR指令的目标寄存器。这是通过将Rt编码为1111来实现的。加载的值被视为地址，执行的效果是该地址的分支。加载值的位[0]选择分支后的执行状态，并且必须具有值1；
* 放弃计算结果；在某些情况下，当一条指令是另一条更一般的指令的特例时，可以这样做，但结果被丢弃；
* 如果LDRB、LDRH、LDRSB或LDRSH指令的目标寄存器说明符为1111，则该指令是内存提示，而不是加载操作；
* 如果MRC指令的目标寄存器说明符为1111，则从协处理器传输的值的位[31:28]被写入APSR中的N、Z、C和V标志，位[27:0]被丢弃；

#### 5.1.3 使用1101作为寄存器说明符

R13在Thumb指令集中定义，主要用作堆栈指针，而R13通常被标识为Thumb中的SP指令。在32位Thumb指令中，如果将SP用作超出本节所述架构定义约束的通用寄存器，则结果是不可预测的。

SP的位[1:0]必须视为SBZP（应为零或保留）。将非零值写入位[1:0]会导致不可预测的行为。读取位[1:0]返回零

SP的32位Thumb指令支持仅限于以下方面：

* SP作为MOV指令的源或目标寄存器。仅支持无移位、无标志置设的寄存器到寄存器传输；
* 通过其对齐倍数向上或向下调整SP；
* SP作为任何加载或存储指令的基寄存器。这支持加载、存储或内存提示指令的基于SP的寻址，具有正或负偏移量，有写回和无写回；
* SP作为任何ADD、CMN、CMP或SUB指令中的第一个操作数Rn；
* SP作为任何LDR或STR指令中的传输寄存器Rt；
* SP作为POP或PUSH指令中的地址；

SP的16位Thumb指令会影响高寄存器的16位数据处理指令。Arm不支持任何其他用途。这会影响CMP和ADD的高寄存器形式，其中Arm不赞成使用SP作为Rm。

### 5.2 16位Thumb指令编码

### 5.3 32位Thumb指令编码

## 6 浮点指令集编码

### 6.1 概述

Armv7-M浮点扩展将浮点指令添加到Thumb指令集。实现此扩展不会影响处理器的操作状态。

### 6.2 浮点指令语法

浮点指令使用Thumb指令集的一般约定。所有浮点指令都以V开头。这将浮点指令集中的指令与Thumb指令区分开来。

1. **数据类型说明符**

dt字段通常包含一个数据类型说明符。

dt字段表示：

* 第二个操作数（如果有）的数据类型；
* 如果没有第二个操作数，则为操作数；
* 结果，当没有操作数寄存器；

1. **寄存器说明符**

dest、src1和src2字段包含寄存器说明符或在某些情况下包含寄存器列表。如果省略dest字段则与src1字段相同。

1. **寄存器列表**

寄存器列表是由逗号分隔并括在括号中的寄存器说明符列表。对寄存器列表中可以显示哪些寄存器有限制。个别说明描述了这些限制。

### 6.3 寄存器编码

FP扩展寄存器为双精度寄存器（64位宽）或者单精度寄存器（32位宽）。

适当时，sz字段第8位将寄存器宽度编码为双精度运算（sz设置为1）或单精度运算。大多数FPv4 SP指令仅为单精度指令，对于这些指令，第8位为0。

### 6.4 浮点数据处理指令

## 7 指令集详解

### 7.1 指令格式

指令说明通常使用以下格式：

* 指令部分标题；
* 指令介绍；
* 具有架构信息的指令编码；
* 汇编程序语法；
* 描述指令操作方式的伪代码；
* 异常信息；
* 注释；

1. **说明部分标题**

指令部分标题给出了该部分所述指令的基本助记符。当一个助记符具有在单独指令部分中描述的多个形式时，后面是括号中形式的简短描述。最常见的用法是区分其中一个操作数是立即数的指令形式和其中一个是寄存器的指令形式。

1. **指令简介**

指令部分标题后面是简要说明该指令特点的文本。本描述不一定完整，也不确定。如果与后面更详细的信息之间存在任何冲突，则后者优先。

1. **指令编码**

编码部分包含一个或多个指令编码的列表。每个Thumb指令编码具有编号标签T1、T2。

指令编码描述包括：

* 关于哪些体系结构变体包括指令的特定编码的信息；
* 一种汇编语法，确保汇编程序选择编码优先于任何其他编码；在某些情况下，给出了多个语法。在其他情况下，可以通过查看汇编程序语法描述并使用它来确定哪个语法对应于正在反汇编的指令来确定要使用的正确语法；
* 编码图；这是16位Thumb编码的一半宽度，而32位Thump编码的全宽度。Thumb编码使用两个半字序列的字节顺序而不是一个字的字节顺序；
* 编码特定伪码；这是一种伪代码，将特定于编码的指令字段转换为后面操作小节中的编码无关伪代码的输入，并在编码中挑选任何特殊情况；

1. **汇编语法**

汇编语法小节描述了指令的标准UAL语法。

语法描述由以下元素组成：

* 使用汇编程序语法原型进行描述，每个原型记录汇编程序代码整行的助记符和操作数部分。当有多条此类线路时，每个原型行被注释以指示编码特定伪代码所需的结果。对于每个指令编码，该信息可用于确定在汇编该语法时是否有任何与该编码匹配的指令可用，如果可用，则哪些指令可用；
* 原型语法行的所有变量或可选字段的描述；一些语法字段跨所有或大多数指令进行标准化；

**汇编程序语法原型行约定：**

* <>：用“<”和“>”括起来的都是用户在该位置提供的值类型的简短描述。这些项通常对应于指令编码图中类似命名的字段。当对应关系仅要求将整数值或寄存器号的二进制编码替换为指令编码时，没有明确描述；
* {}：由“{”和“}”括起来的任何项都是可选的。后续文本通常会提供该项的说明以及该项的存在或不存在如何在指令中进行编码。许多指令具有可选的目标寄存器。除非另有说明，如果省略了这样的目标寄存器，则其与指令语法中紧跟其后的源寄存器相同；
* 空格：单个空格用于分隔项目。当一个空格在汇编程序语法中是必需的时，使用两个或多个连续的空格；
* 正负号：这表示可选的正号或负号；如果两者都未编码，则假定为+符号；

1. **异常信息**

处理器异常如下：

* 重置和中断，包括NMI、PendSV和SysTick等；可能发生在任何指令执行之前或之后，在某些情况下，也可能发生在指令执行期间，但通常不是由相关指令引起的；
* 针对执行显式数据存储器访问的所有指令，列出了总线忙异常；
* 使用异常可能因各种原因而发生，并根据说明列出；
* SVC指令中触发SVCall异常；
* BKPT指令触发调试异常；

### 7.2 标准汇编程序语法字段

所有或大多数指令的标准字段：

* ＜c＞是可选字段：指定执行指令的条件。如果省略<c>，则默认为始终（AL）；
* <q> 指定指令上的可选汇编程序限定符。N表示窄，指定汇编程序必须为指令选择16位编码。W表示宽，指定汇编程序必须为指令选择32位编码。如果未指定W或N，则汇编程序可以选择16位或32位编码。如果两者都可用，则必须选择16位编码；

### 7.3 条件执行

Armv7-M中的大多数Thumb指令可以根据APSR条件标志的值有条件地执行。在Thumb指令中，条件（不是AL）通常在前一条IT指令中编码。一些条件分支指令不需要前面的IT指令，并且在其编码中包括条件代码。

### 7.4 寄存器的移位

### 7.5 内存访问

内存访问指令通常允许以下寻址模式：

* 将偏移值添加到从基寄存器获得的地址或从中减去。结果用作存储器访问的地址。基本寄存器不变。汇编语言语法为[<Rn>，<offset>]；
* 预索引寻址偏移值应用于从基寄存器获得的地址。结果用作内存访问的地址，并写回基寄存器。此模式的汇编语言语法为[<Rn>，<offset>]!；
* 后索引寻址从基寄存器获得的地址被用作内存访问的地址，不改变。偏移值应用于地址，并写回基寄存器。汇编语言语法为[<Rn>]，<offset>；

在每种情况下，<Rn>是基寄存器，<offset>可以是立即常数、索引寄存器或移位索引寄存器。

### 7.6 提示指令

Thumb指令集包括以下类型的提示指令：内存提示和NOP兼容提示指令。

1. **内存提示指令**

一些Rt设置为1111的加载指令是内存提示指令。内存提示能够向内存系统提供有关未来内存访问的高级信息，而无需实际加载或存储任何数据。PLD和PLI是当前定义的唯一内存提示指令。

1. **NOP兼容提示指令**

与内存访问无关的提示指令属于单独的提示指令类别，称为NOP兼容的提示指令。NOP兼容提示指令提供实现定义的行为或作为NOP。

### 7.7 Armv7-M Thumb指令列表

## 8 系统级程序模型

### 8.1 系统级简介

Arm架构定义了软件操作的层次结构：

* 最低级别是应用程序级别， 应用程序级软件在很大程度上独立于架构；
* 更高级别是系统级，包括对应用程序的支持。在不同的Armv7体系结构中，系统级功能及其支持方式有很大的不同；

系统级支持需要特权访问，为系统软件提供配置和控制资源所需的访问权限。通常，操作系统提供这种控制，以透明的方式或通过应用程序发起的系统调用向应用程序提供系统服务。操作系统还负责服务中断和其他系统事件，使异常成为系统级程序模型的关键组成。

### 8.2 Armv7-M内存映射架构

Armv7-M是一种内存映射架构，这意味着该架构为处理器分配中断向量和系统控制和配置的物理地址。

Armv7-M在表中维护异常处理程序入口地址。体系结构保留地址空间0xE0000000到0xFFFFFFFF供系统级使用。Arm保留该系统地址空间的第一个1MB，0xE0000000到0xE00FFFFF作为专用外围总线（PPB）。从0xE0100000开始的其余地址空间的分配是实现定义的，带有一些内存属性限制。

在PPB地址空间中，架构将4KB内存块（从0xE000E000到0xE000EFFF）作为系统控制空间（SCS）。SCS用作处理器ID寄存器、一般控制和配置（包括向量表基址）、系统处理程序支持（用于系统中断和异常）、系统定时器等。在Armv7-M架构中，所有异常和中断，包括由NVIC处理的外部中断，共享一个由SCS中的寄存器控制的公共优先级模型。

### 8.3 系统级术语

#### 8.3.1 模式、特权和堆栈

**模式**：M系列处理器支持两种操作模式：线程模式和处理模式。线程模式在重置时进入并且从异常返回时也会进入。当处理器处理异常时自动进入处理模式。处理器必须处于处理模式才能从异常返回中返回。

**特权**：代码可以作为特权或非特权执行。非特权执行限制或排除对某些资源的访问。特权执行可以访问所有资源。处理模式下的执行始终具有特权。线程模式下的执行可以是特权的，也可以是非特权的。

**堆栈指针**：处理器实现一对存储体堆栈指针：主堆栈指针和进程堆栈指针。在处理模式下，处理器只能使用主堆栈指针。在线程模式下可以使用任一堆栈指针。

#### 8.3.2 异常

异常是改变程序中正常控制流的情况。异常行为分为两个阶段：①异常生成，异常事件发生并呈现给处理器；②异常处理：处理器遵循异常生成、异常处理程序代码执行和异常返回序列时，从异常生成到异常处理的转换可以是瞬时的。

Armv7-M定义了以下异常类别：

* 重置：重置是一种特殊形式的异常；当触发时，以可能不可恢复的方式终止当前执行。当重置被取消时，执行从固定点重新开始；
* 系统调用（SVCall）：由SVC指令显式导致的异常；应用软件使用SVC指令调用底层操作系统。SVC指令使应用程序能够发出要求对系统进行特权访问的系统调用，并按照程序顺序执行。Armv7-M还支持中断驱动的监控器调用机制PendSV；
* 故障：故障是由指令执行中的错误条件导致的异常。故障可以同步或异步报告给导致故障的指令。通常，故障是同步报告的。不精确总线故障是Armv7-M中支持的异步故障；同步故障总是与导致故障的指令一起报告。体系结构不能保证异步故障相对于导致故障的指令如何报告；
* 中断：中断是一种异常，而不是重置、故障或系统调用。所有中断都与指令流异步。通常中断由系统中必须与处理器通信的其他组件使用；

每个异常都有异常编号、优先级别以及内存中的一个向量（定义了异常执行的入口点)。向量中的值是对应异常的异常处理程序或中断服务例程（ISR）入口点的地址。

除重置外，异常具有以下可能状态：

* 非活动：未挂起或活动的异常；
* 挂起：已生成但处理器尚未开始处理的异常。当相应的异常事件发生时，将生成异常；
* 激活：处理器已开始执行相应异常处理程序但尚未从该处理程序返回的异常。异常的处理程序正在运行或者被优先级更高的异常处理程序抢占；
* 活动和挂起：异常的一个实例处于活动状态，异常的第二个实例处于挂起状态。只有异步异常可以是活动和挂起的；

异常优先级决定处理器处理异常的顺序：

* 每个异常都有一个优先级，即异常优先级。三个异常具有固定优先级，而其他都具有可由特权软件配置的优先级；
* 在处理器上执行的指令流具有与其相关的优先级，即执行优先级；
* 重置后执行优先级是执行优先级的最低级别。只有线程模式下的执行才能处于此执行优先级；
* 异常优先级充分高于执行优先级的异常转变为活动状态；

软件可以使用寄存器提高执行优先级，否则执行优先级是所有活动异常中的最高优先级。当异常由于其优先级高于执行优先级而变得活动时其异常处理程序抢占当前运行的指令流并且其优先级成为执行优先级。

当重置以外的异常抢占指令流时，处理器自动将关键上下文信息保存到堆栈中，执行分支到相应异常向量所指向的代码。异常可以在异常激活期间发生，例如由于保存上下文信息而产生内存故障。此外，该架构允许优化延迟到达的异常。

处理器始终以处理程序模式运行异常处理程序。如果异常抢占以线程模式运行的软件，则处理器将作为异常条目的一部分更改为处理程序模式。处理器以处理程序模式执行异常处理程序，并从处理程序返回。

异常返回时：

* 如果异常状态为活动和挂起：如果异常具有足够的优先级，将变为活动状态，处理器将重新进入异常处理程序；否则，将变为挂起；
* 如果异常状态为活动状态，则变为非活动状态；
* 处理器恢复其在异常条目上保存的上下文信息；
* 如果被异常处理程序抢占的代码在线程模式下运行，处理器将更改为线程模式；
* 处理器恢复执行被异常处理程序抢占的代码；

异常返回链接是存储在异常条目链接寄存器中的值，确定异常返回的目标。

### 8.4 寄存器

Armv7-M具有与处理器紧密联系的以下寄存器：

* 通用寄存器R0-R12；
* 两个堆栈指针（SP）寄存器；
* 链路寄存器LR，也称为R14；
* 程序计数器（PC），有时称为R15；
* 状态寄存器用于标志、执行状态位以及处理异常时的异常编号；
* 用于管理异常和中断优先级方案的屏蔽寄存器；
* 一个控制寄存器，用于标识当前堆栈和线程模式特权级别；

#### 8.4.1 核心寄存器

寄存器R0-R12、SP、LR和PC被称为Arm核心寄存器。这些寄存器可以描述为R0-R15。Armv7-M处理器实现两个堆栈主堆栈（MSP）和进程堆栈（PSP）。当前堆栈取决于模式，在线程模式下，取决于CONTROL.SPSEL位。重置选择并初始化MSP。

Armv7-M实现将SP位[1:0]视为RAZ/WI。Arm强烈建议软件将SP位[1:0]视为SBZP，以实现Armv7最大可移植性。

。

#### 8.4.2 专用程序寄存器

**应用程序状态寄存器（APSR）**：保存可由应用程序级软件（即非特权软件）写入的标志。MSR和MRS指令对应用程序级可写标志的处理在所有Armv7中都是一致的。标志设置指令修改APSR标志N、Z、C、V和Q，处理器使用这些标志评估和条件分支指令中的条件执行。

**中断程序状态寄存器（IPSR）**：当处理器执行异常处理程序时，保存正在处理异常的异常号，否则IPSR值为零。处理器在异常进入和退出时写入IPSR。软件可以使用MRS指令读取IPSR，但处理器忽略MSR指令对IPSR的写入。IPSR异常数字段在线程模式下值为0。

**执行程序状态寄存器（EPSR）**：保存执行状态位；EPSR包含T位，设置为1表示处理器执行Thumb指令。使用MRS指令将所有字段读取为零，并且处理器忽略MSR指令对EPSR的写入。

#### 8.4.3 专用掩码寄存器

**异常掩码寄存器（PRIMASK）**：一个1位寄存器；将PRIMASK设置为1会将执行优先级提高到0。非特权访问是RAZ/WI；

**基本优先级掩码（BASEPRI）**：一个8位寄存器；BASEPRI更改异常抢占所需的优先级。只有当BASEPRI的值低于当前执行软件的未屏蔽优先级时才有效。值为零将禁用BASEPRI的屏蔽。非特权访问是RAZ/WI；

**故障掩码故障掩码（FAULTMASK）**：一个1位寄存器；将FAULTMASK设置为1会将执行优先级提高到-1，即硬件故障的优先级。只有优先级低于-1的特权软件才能将FAULTMASK设置为1。这意味着硬故障和NMI处理程序不能将FAULTMASK设置为0；

#### 8.4.4 专用控制寄存器

专用控制寄存器是一个2位或3位寄存器：

* nPRIV：第0位，定义线程模式下的执行权限；
* SPSEL：第1位，定义要使用的堆栈；
* FPCA：第2位，使能处理器fp扩展；

### 8.5 Armv7-M异常模型

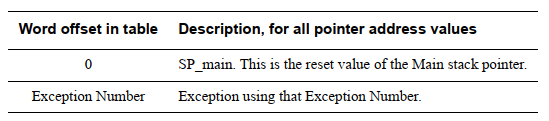
#### 8.5.1 支持的异常概述

Armv7-M支持重置、NMI、硬件异常、内存管理异常、总线异常、使用异常、调试监视器异常、管理调用以及和496个外部中断。

异常号1到15系统架构预定义的异常，16到N为外部中断。

#### 8.5.2 异常向量表

向量表包含堆栈指针的初始化值，以及每个异常处理程序的入口点地址。异常号为0对应于主堆栈指针。



在重置时，处理器将向量表的基址初始化为一个实现定义的地址。软件可以找到表的当前位置，或使用VTOR重新定位表。所有异常条目都必须将第0位设置为1，因为该位定义了进入异常时EPSR中T位。

向量表必须自然对齐到2的幂次方字节，最小对齐长度为128字节。在启动或复位时，处理器使用偏移量0处的地址作为主堆栈地址的初始值。

#### 8.5.3 异常优先级和抢占

在Armv7-M优先级模型中，较低的数字优先运行。分配的优先级值越低，优先级别越高。具有相同优先级的异常的优先级顺序是固定的，并由异常号决定。Reset、NMI和HardFault分别以固定的优先级-3、-2和-1执行。软件可以使用系统控制空间中的寄存器来设置所有其他异常的优先级。

当多个挂起异常具有相同的优先级编号时，异常编号最低的异常优先。当一个异常是活动的，只有具有更高优先级的异常才能抢占。

优先级分组将异常优先级分为两部分：组优先级和子优先级。AIRCR中PRIGROUP字段通过指示8位优先级字段中的多少位为子优先级来控制这种分组。

组优先级字段定义抢占的优先级。如果多个挂起的异常具有相同的组优先级，则异常处理逻辑使用子优先级字段来解析组内的优先级。Reset、NMI和HardFault的组优先级分别固定为-3、-2和-1。

执行优先级意味着异常处理程序的执行优先级可以高于相应异常的优先级。特别是，如果处理程序降低了相应异常的优先级，则执行优先级只会落在优先级最高的被抢占异常的优先级上。

优先级分组将异常优先级分为组优先级和子优先级两部分。AIRCR寄存器中PRIGROUP字段通过指示8位优先级字段中有多少位指定子优先级来控制这种分割。组优先级字段定义了抢占的优先级。如果多个挂起的异常具有相同的组优先级，则异常处理逻辑使用子优先级字段来解析组内的优先级。复位、NMI和硬件错误的组优先级分别为-3、-2和-1。

将PRIMASK设置为1将使执行优先级为0。这可以防止任何具有可配置优先级的异常变为活动状态。

将FAULTMASK此设置为1，则执行优先级为-1。只有当执行优先级不为NMI或硬件错误时，软件才能将FAULTMASK设置为1，即只有当优先级大于等于0时，FAULTMASK才能设置为1。设置FAULTMASK将异常处理程序的优先级提升到硬件错误级别。除NMI外的任何异常返回都会自动将FAULTMASK清除为0。

优先级提升机制只影响组的优先级。对次优先级没有影响。子优先级仅用于对挂起的异常优先级进行排序，不影响活动异常。

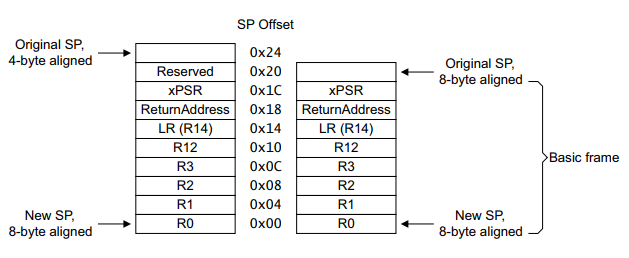
#### 8.5.4 进入异常

在抢占指令流时，硬件将上下文状态保存到一个由SP寄存器指向的堆栈上。使用的堆栈取决于异常发生时处理器的模式。栈中上下文支持Arm架构过程调用标准(AAPCS)。这意味着异常处理程序可以是符合AAPCS的过程。Armv7-M架构使用全降序堆栈。

当将上下文压入堆栈时，硬件保存8个32位的单词，包括xPSR、返回地址、LR(R14)、R12、R3、R2、R1和R0。

#### 8.5.5 异常进入时的堆栈对齐

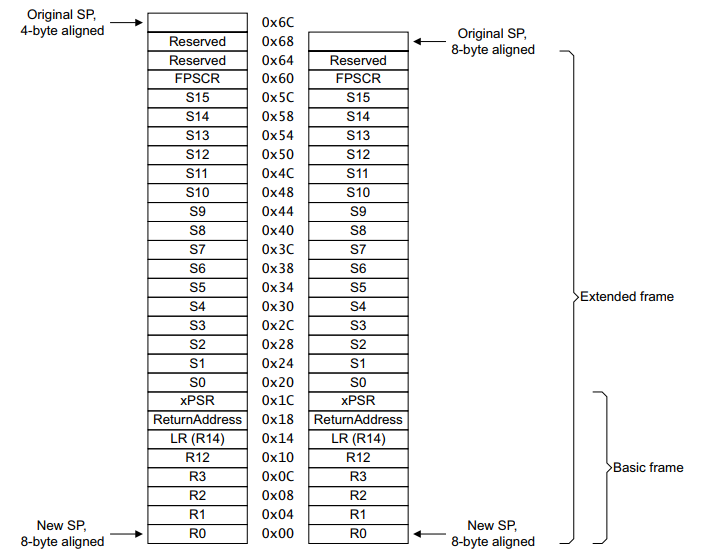
Armv7-M体系结构保证堆栈指针值至少是4字节对齐的。然而，一些软件要求堆栈指针是8字节对齐的，并且体系结构可以强制这种对齐。CCR中STKALIGN位指示是否1堆栈指针进行强制8字节对齐。



因为异常可能发生在任何指令边界上，所以当处理器发生异常时，当前堆栈指针可能不是8字节对齐的。Arm建议将异常处理程序编写为符合AAPCS的函数，并且AAPCS在进入符合AAPCS的函数时需要8字节的堆栈指针对齐。

当CCR中STKALIGN设置为1时，确保在异常进入之前使用的堆栈指针具有8字节对齐方式。如果需要可以调整其对齐方式。当处理器将PSR值推入堆栈时，使用堆栈上PSR值的第9位来指示是否重新排列了堆栈。

在异常返回时，当CCR中STKALIGN设置为1时，处理器使用从堆栈中弹出的PSR值的第9位的值来确定是否必须调整堆栈指针的对齐方式。



任何异常处理代码必须从堆栈检索参数(在发生异常之前被压入堆栈)，必须使用xPSR中第9位来确定前一个堆栈顶部的偏移量是0x20还是0x24。如果实现包含FP扩展，那么这样的代码必须使用xPSR中第9位和EXC\_RETURN中第4位的值，以确定之前的堆栈顶部是否位于偏移量0x20、0x24、0x68或0x6c。

#### 8.5.6 异常返回

当处理器处于处理模式，并且通过指令将值0xFXXX加载到PC中时，处理器便会从异常中返回。当以这种方式使用时，处理器拦截写入PC的值。

EXC\_RETURN异常返回值：

* 第31到28位： 0xf；这个值将PC加载中的值标识为EXC\_RETURN值；
* 第27到5位：保留，SBOP；将除1以外的值写入该字段中的任何位是不可预测的；
* 第4位：如果处理器没有实现FP扩展则保留，SBOP；如果处理器实现了FP扩展，定义此异常的堆栈帧是否为FP状态信息分配了空间；
* 第3到0位：定义异常返回行为；

如果在线程模式下，或者从向量表中，或者通过任何其他指令将EXC\_RETURN值加载到PC中，则该值将被视为地址，而不是特殊值。0xFXXXXXXX地址范围(包括所有可能的EXC\_RETURN值)具有不可执行权限，加载该值会导致内存管理异常

异常或使用异常或将异常升级为硬件错误。

Armv7-M体系结构对异常返回提供了许多完整性检查。这可以防止系统软件中的错误。错误的异常返回信息可能与处理器在硬件中保存的执行状态不一致，或者与异常机制存储的其他状态不一致。与硬件相关的完整性检查确保跟踪NVIC和SCB硬件中的活动异常与异常返回一致。

异常返回完整性检查：

* 在返回开始时，IPSR中保存的异常编号在SCB中为活动状态；
* 正常情况下，如果除正在返回的异常外，至少有一个异常处于活动状态，则必须将返回值设置为处理程序模式；
* 在返回线程模式时，恢复到IPSR异常号字段的值必须为0；
* 在返回处理模式时，恢复到IPSR异常号字段的值不能为0；
* EXC\_RETURN[3:0]不能是保留值；

任何失败的检查都会导致使用错误异常，其中的EXC\_RETURN值在LR中。当硬件错误处于活动状态而NMI处于非活动状态时，异常返回总是使硬件错误处于非活动状态并清除FAULTMASK。

#### 8.5.7 异常进入时发生异常

在异常输入期间，可能会发生其他异常。这要么是因为异常输入中涉及的操作发生了错误，要么是因为异步异常(即中断)的到来，该中断的优先级高于当前进入的异常。

Armv7-M架构没有指定异常入口中处理器识别异步异常到达的点。然而，为了支持非常低的中断延迟，该体系结构允许在异常进入期间到达的高优先级中断在异常进入期间变为活动的。当处理器在异常输入期间接受异步中断时，导致异常进入的异常称为原始异常。由中断引起的异常称为迟到异常。

在这种情况下，由原始异常启动的进入异常可以被延迟到达的异常使用。处理器在从延迟到达的异常返回后获取原始异常。这被称为迟发抢占。

#### 8.5.8 异常返回时发生异常

在异常返回期间，其他异常可能会影响行为，这要么是因为异常返回期间执行的操作出现了错误，要么是因为异步异常的优先级高于异常返回的优先级。异步异常可能已经挂起，或者可能在异常返回期间到达。

异常返回链接描述了异常返回的目标。目标优先级为：优先级最高的活动异常的优先级，不包括正在返回的异常；由专用掩码寄存器设置的增强优先级。

### 8.6 浮点支持

#### 8.6.1 启用浮点支持

如果Armv7-M实现包括FP扩展，则使用该扩展的任何系统的引导软件必须确保在协处理器访问控制寄存器中启用对CP10和CP11的访问。如果不这样做，则FP功能的操作未定义。如果对CP10和CP11的访问控制位编程不同，浮点特征的操作是不可预测的。

#### 8.6.2 FP扩展系统寄存器

在Armv7-M实现中，FP扩展在CP10和CP11寄存器空间中有一个系统寄存器。任何包含此扩展的Armv7-M实现都必须实现此寄存器。

## 9 系统内存模型

### 9.1 缓存和分支预测

#### 9.1.1 缓存标识

Armv7缓存标识由一组寄存器组成，这些寄存器描述了处理器控制下实现的缓存：

* 单个缓存类型寄存器定义：任何指令缓存的最小行长度、任何数据或统一缓存的最小行长度以及一级指令缓存的缓存索引和标记策略；
* 单个缓存级别ID寄存器定义：-每个缓存级别实现的缓存类型、缓存的一致性级别以及缓存的统一级别；
* 单个高速缓存大小选择寄存器设置当前高速缓存大小并且标识寄存器的高速缓存级别和高速缓存类型；
* 对于每个实现的缓存，在所有级别的缓存中，缓存大小标识寄存器定义：-缓存是否支持直写、回写、读分配和写分配；

在Armv7中，体系结构定义了对多级缓存的支持，最多支持七级。这使得识别Armv7处理器可用的缓存资源的过程复杂化。

获取缓存信息：

* 读取缓存类型寄存器以查找用于1级指令缓存的索引和标记策略；该寄存器还提供用于指令缓存、数据缓存和统一缓存的最小缓存线的大小；
* 读取缓存级别ID寄存器以查找实现了哪些缓存。寄存器包括七个缓存类型字段，用于缓存级别1到7。从级别1开始扫描这些字段，识别在每个级别实现的指令、数据或统一缓存。当扫描达到未定义缓存的级别时，此扫描结束；
* 对于第2阶段确定的每个缓存，写入缓存大小选择寄存器以选择所需缓存以及读取缓存大小ID寄存器以查找缓存的详细信息；

#### 9.1.2 缓存启用和禁用

在Armv7-M中，架构定义了对多级缓存的控制。在Armv7-M中，配置和控制寄存器CCR用于启用和禁用缓存。CCR中DC位启用或禁用处理器可见的所有级别缓存中的所有数据和统一缓存。CCR中IC位启用或禁用处理器可见的各级缓存中的所有指令缓存。

如果某个实现需要对缓存启用进行更细粒度的控制，可以在辅助控制寄存器中为此实现控制位。例如，实现可能定义控制位来启用和禁用特定级别的缓存。

#### 9.1.3 缓存行为

当内存位置具有缓存属性时，是否保存在缓存中仍然取决于实现的许多方面。

* 缓存的大小、行长度和关联性；
* 缓存分配算法；
* 系统中可以访问内存的其他模块的活动；
* 中断行为；

缓存的行为：

* 如果禁用了缓存，则可以保证不会从缓存分配新的位置；
* 如果启用了缓存，则保证没有向缓存分配不具有可缓存属性的内存位置；
* 如果启用了缓存，如果该位置的访问权限定义为不能通过读访问和不能通过写访问，则可以保证没有分配缓存；
* 可以覆盖的内存的最大大小称为缓存回写粒度；
* 在某些实现中，CTR标识了缓存回写的粒度；

出于这些原因，一个缓存条目至少覆盖16字节，不超过2KB的连续地址空间，与它的大小保持一致。

**重启时缓存行为**：

* 在重置时，所有缓存都被禁用。
* 实现可能需要使用特定的缓存初始化例程，在启用存储阵列之前使其无效。任何必需的初始化例程的确切形式都是由实现定义的；
* 在缓存禁用时，由实现定义访问是否可以生成缓存命中；
* 如果某个实现允许缓存命中，而缓存内容在缓存禁用时没有失效，则初始化例程必须避免从未初始化的缓存运行的任何可能性；

#### 9.1.4 分支预测

分支预测硬件通常使用高速缓存的形式来保存分支信息。Armv7-M架构要求该分支预测硬件对软件不可见。

在Armv7-M中，刷新分支预测器之后不需要使用分支预测维护操作来使分支预测无效。

#### 9.1.5 用于描述缓存维护操作的术语

#### 9.1.6 Armv7-M缓存层次结构的抽象

#### 9.1.7 缓存和分支预测器维护操作

#### 9.1.8 系统级缓存

系统级体系结构可以定义未由Armv7-M体系结构定义的缓存和内存模型的软件视图的其他方面。系统级架构的这些方面可能会影响缓存和一致性的软件管理需求。例如，系统设计可能会引入额外的缓存级别，这些级别无法使用Armv7-M体系结构定义的维护操作进行管理。这种缓存被称为系统缓存并通过使用内存映射操作进行管理。Armv7-M体系结构不禁止存在体系结构范围之外的系统缓存。

### 9.2 一般内存系统操作的伪代码

## 10 系统内存映射

### 10.1 系统地址映射

Armv7-M支持预定义的32位地址空间，对代码、数据和外围设备进行细分，以及片上和片外资源区域，其中片上资源指与处理器紧密联系的内存资源。地址空间支持八个0.5GB的主分区分别为代码、SRAM、外围设备、两个RAM区域、两个设备区域和系统。

体系结构分配物理地址，用作事件入口点（向量）、系统控制和配置。事件入口点都是相对于表基地址定义的，表基地址在重置时配置为实现定义的值，然后保持在为系统配置和控制保留的地址空间中。为了满足系统需求，保留地址空间0xE0000000到0xFFFFFFFF供系统级使用。

支持识别非特权访问和特权访问的软件模型需要内存保护方案来控制访问权限。内存保护系统架构(PMSAv7)是一个可选的系统级特性。

地址范围为0xE0000000至0xE0100000的PPB的规则如下：

* 该区域定义为强有序内存；
* 无论处理器的端序状态如何，寄存器访问始终是小端序；
* 一般而言，除非另有说明，寄存器仅支持字访问，字节和半字访问不可预测；
* 保留寄存器或位字段必须视为UNK/SBZP；
* 有关调试相关资源；

除非另有说明，否则对PPB的非特权访问将导致总线故障。

### 10.2 系统控制空间（SCS）

系统控制空间（SCS）是一个内存映射的4KB地址空间，为配置、状态报告和控制提供32位寄存器。SCS寄存器分为以下几组系统控制和识别、CPUID处理器标识空间、系统配置和状态、故障报告、系统定时器、嵌套V扇区中断控制器（NVIC）、受保护内存系统架构（PMSA）和系统调试。

### 10.3 系统定时器(SysTick)

Armv7-M实现必须包括一个系统定时器SysTick，提供了一个简单的24位写时清零、递减、计数器和灵活的控制机制。系统可以通过多种不同方式使用此计数器。

定时器由四个寄存器组成：

* 一个控制和状态寄存器；配置时钟、启用计数器、启用中断并指示计数器状态；
* 计数器重载寄存器；
* 计数器当前值寄存器；
* 校准值寄存器；

启用时，计时器从SYST\_CVR中的值开始倒数。当计数器达到零时，在下一个时钟边沿重新加载SYSTURVR中的数值。然后在后续时钟上递减。计数器达到零时的这种重新加载称为回环。当计数器为零时，将COUNTFLAG状态位设置为1。读取COUNTFLAG的状态位将其清除为0。

写入SYST\_CVR将寄存器和计数标志状态位都清除为零。这导致定时器逻辑在下一个定时器时钟上从SYST\_RVR重新加载SYST\_CVR。写入SYST\_CVR不会触发异常逻辑。读取SYST\_CVR返回访问寄存器时计数器的值。将值0写入SYST\_RVR将禁用下一次计数器。

计时器由参考时钟计时。参考时钟是处理器时钟还是外部时钟源由具体实现定义。如果实现使用外部时钟，则必须记录处理器时钟和外部参考之间的关系。考虑亚稳态、时钟偏斜和抖动，这是系统定时校准所必需的。

### 10.4 嵌套中断控制器（NVIC）

Armv7-M提供了一个中断控制器，作为Armv7-M异常模型的组成部分。中断控制器操作符合Arm通用中断控制器（GIC）规范。Armv7-M NVIC架构支持多达496个中断。支持的外部中断的数量可以从系统控制空间中地址0xE000E004处访问的中断控制器类型寄存器（ICTR）中确定。

NVIC操作Armv7-M支持电平敏感和脉冲敏感中断行为。这意味着可以处理电平触发和脉冲触发中断。脉冲触发的中断源必须保持足够长的时间，以便由处理器时钟可靠地采样，以确保被锁存并成为挂起状态。后续脉冲可将挂起状态添加到活动中断，使中断状态为活动和挂起。然而在活动期间发生的多个脉冲仅注册为中断调度的单个事件。

作为Armv7-M异常模型及其优先级策略的一部分，所有NVIC中断都具有可编程优先级值和相关异常编号。

NVIC支持以下功能：

* NVIC中断可以通过写入相应的中断使能寄存器相应字段来启用和禁用中断。寄存器使用写1使能和写1清除策略。当中断被禁用时，中断触发会导致中断变为挂起状态，但中断不能变为活动状态。如果中断在禁用时处于活动状态，将保持活动状态，直到重置或异常返回清除；
* 软件可以使用一对互补的寄存器（设置挂起寄存器和清除挂起寄存器）设置或清除NVIC中断的挂起状态；
* 提供活动位状态，使软件能够确定中断是处于非活动、活动、挂起还是活动和挂起状态；
* 通过更新32位寄存器中的8位字段（每个寄存器支持四个中断），对NVIC中断进行优先级排序；

### 10.5 保护内存系统架（PMSAv7）

支持非特权和特权软件的模型需要控制访问权限的内存保护方案。Armv7-M支持内存保护系统架构（PMSAv7）。PMSAv7实现的系统地址空间由存储器保护单元（MPU）保护。MPU将存储器划分为多个区域。支持区域的数量由实现定义。PMSAv7可以支持小到32字节的区域，但是4GB地址空间中有限的寄存器资源意味着MPU提供了固有的粗粒度保护方案。

#### 10.5.1 MPU与系统内存映射的关系

当实现时，MPU与系统地址映射中描述的系统内存映射的关系如下：

* MPU支持提供对物理地址访问权限的控制，不执行地址转换；
* 当MPU被禁用或不存在时，系统采用默认系统内存映射。

当MPU启用时，启用区域定义系统地址映射：

* 对专用外围总线（PPB）的访问始终使用默认系统地址映射；
* 从向量地址表读取的异常向量始终使用默认系统地址映射；
* MPU在如何更改与系统空间（即地址0xE0000000及更高）相关的默认内存映射属性方面受到限制；
* 当执行优先级小于0时，MPU\_CTRL中HFNMIENA位确定内存访问是使用MPU还是默认内存映射属性。如果处理器正在执行NMI或硬故障处理程序，或者如果FAULTMASK设置为1，则执行优先级小于0；
* 默认系统内存映射可以配置为提供特权访问的背景区域；
* 在多个区域中地址匹配的访问使用访问属性的最高匹配区域号；

#### 10.5.2 MPU被禁用时的行为

通过设置MPU\_CTRL寄存器中ENABLE来禁用MPU。启用位为0，表示特权和非特权访问使用默认内存映射。

当MPU被禁用时：

* 指令访问使用默认内存映射和属性；
* 数据访问使用默认内存映射和属性;不执行内存访问权限检查，也不会产生中止；
* 程序流预测功能正常，由CCR中BP位控制；
* 根据默认内存映射；

#### 10.5.3 符合PMSAv7的MPU操作

Armv7-M仅支持与MPU区域支持相关的统一内存模型。所有启用的区域都支持指令和数据访问。区域的基址、大小和属性都是可配置的，一般规则是所有区域自然对齐。如果两个区域之间存在重叠，则具有最高区域编号的寄存器优先。

子区域支持256字节或更大的区域，该区域可分为大小为2（N-3）的八个子区域。一个区域内的子区域可以针对相关区域属性寄存器单独禁用（8个禁用位）。禁用子区域时，需要从另一个区域进行访问匹配，如果启用，则需要进行背景匹配。如果访问匹配未发生，则生成故障。小于256字节的区域大小不支持子区域，设置为MPU\_RASR。小于256字节的区域的SRD为非零是不可预测的。

#### 10.5.4 在SCS中对PMSAv7的寄存器

通常，除非另有说明，寄存器仅支持字访问，字节和半字访问不可预测。所有MPU寄存器地址都映射为小端。MPU寄存器需要特权内存访问才能进行读写。任何未经授权的访问都会产生总线异常。

有三个通用MPU寄存器：

* MPU类型寄存器中指定MPU类型，该寄存器可用于确定MPU是否存在以及支持的区域数量；
* MPU控制寄存器MPU\_CTRL中指定MPU控制器寄存器；MPU控制器包括一个全局启用位，必须设置为1才能启用MPU；
* MPU区域编号寄存器MPU\_RNR中规定的MPU区域号寄存器；

MPU区域编号寄存器选择相关区域寄存器：

* MPU区域基址寄存器MPU\_RBAR中指定MPU区域基地址寄存器；
* MPU区域属性和大小寄存器，用于控制区域大小、子区域访问、访问权限、内存类型和内存区域的其他属性；

每组区域寄存器包括其自己的区域使能位。如果Armv7-M实现不支持PMSAv7，则只需要MPU类型寄存器。MPU控制寄存器为RAZ/WI，该区域中的所有其他寄存器均保留为UNK/SBZP。所有MPU寄存器均为32位宽。

## 11 CPUID方案

### 11.1 CPUID方案概述

CPUID寄存器组描述了Arm处理器实现的功能，为Armv7体系结构定义了一组通用寄存器。

在主ID寄存器中指定0xF的架构变量表示使用CPUID方案。在Armv7-M配置中主ID寄存器称为CPUID基本寄存器。在Armv7-M配置文件中，基本体系结构和任何实现的体系结构扩展定义了CPUID字段值。

除了CSSELR之外，CPUID寄存器是只读的并且只有特权访问。处理器忽略特权写入，任何非特权数据访问都会导致总线异常。CSSELR只能由特权软件访问并且支持读写。

Armv7-M中的CPUID选项是Armv7-A和Armv7-R中CPUID的子集，因此寄存器中的某些字段在Armv7-M不能有值。

**Armv7-M保留字段：**在Armv7-M中保留，但在其他配置文件中定义。

**保留字段**：在所有Armv7配置文件中保留。

保留字段和Armv7-M保留字段被读取为零（RAZ）。在任何字段描述中，未列出的任何字段值都是保留的。如果字段的可能值显示为Armv7-M保留，则意味着Armv7-M实现无法使用该字段值。

### 11.2 处理器功能ID寄存器

### 11.3 调试功能ID寄存器

### 11.4 辅助特征ID寄存器

### 11.5 内存模型特征寄存器

### 11.6 指令集属性寄存器

### 11.7 浮点特征识别寄存器

### 11.8 缓存控制标识寄存器

## 12 系统指令

### 12.1 Armv7-M系统指令概述

Armv7-M仅在Thumb状态下执行指令。为了支持在软件控制下读写专用寄存器，Armv7-M提供了三个系统指令：CPS、MRS和MSR。Armv7-M系统指令中使用的特殊寄存器编码描述了用于MSR和MRS指令参数的编码。

### 12.2 Armv7-M系统指令

#### 12.2.1 CPS

#### 12.2.2 MRS

#### 12.2.3 MSR